```
POLYNOM_1
---------
*C polynom_1_begin
               ************ POLYNOM_1 ************
               This theory contains the definitions of classes for constructing
               polynomial algebras.
               The classes are:
               1. Free abelian monoids:
                  The monomials on a set of indeterminates can be characterized
                  as a free abelian monoid.
               2. Monoid copower:
                  This class is regarded as a core class for both monomials
                  and polynomials. Any monoid copower construction can be specialized
                  to give a free abelian monoid, and generalized to give a free
                  monoid algebra.
               3. Free monoid algebras:
                  The polynomials over a monoid of monomials and a commutative ring
                  of coefficients can be characterized as a free monoid algebra.
               4. Polynomial algebras:
                  Elements of this class can be regarded as full implementations
                  of polynomials.
               These standard constructions are described in most algebra books
               (e.g. Lang).
*C polynom_1_summary
               Class definitions for free abelian monoids,
               free abelian groups, monoid copowers,
               free monoid algebras and polynomial algebras.
*C polynom_1_intro
               Class definitions given for free abelian monoids,
               free abelian groups, monoid copowers,
               and free monoid algebras. These classes are used
               as ADT's for functions of finite support with various
               domains and ranges.
               A class definition for polynomial algebras
               serves as an ADT for
               implementations of multivariate polynomials.
*C free_abmonoid_com
               =========================
               FREE ABELIAN MONOID CLASS
               =========================
*D free_abmonoid_df
               FAbMon{<i:level>}(<S:S:*>)== free_abmonoid{<i>:1}(<S>)
               FAbMon(<S:S:*>)== free_abmonoid{i:1}(<S>)
*A free_abmonoid
               FAbMon(S) ==
                 mon:AbMon
                 × inj:(|S| ⟶ |mon|)
                 × (mon':AbMon ⟶ f':(|S| ⟶ |mon'|) ⟶ {!g:MonHom(mon,mon') | g o inj = f'})
*T free_abmonoid_wf  4.9 sec.
⊢ ∀S:DSet. FAbMon(S) ∈ 𝕌'
|
BY Unfold `free_abmonoid` 0 THEN Auto
*D free_abmon_mon_df          <f:free_abmon:E>.mon== free_abmon_mon{}(<f>)
*A free_abmon_mon             f.mon ==  f.1
*T free_abmon_mon_wf  0.4 sec.
⊢ ∀S:DSet. ∀f:FAbMon(S).  f.mon ∈ AbMon
|
```

```
BY ModulePiTac 3 ‘‘free_abmon_mon free_abmon_inj free_abmon_umap‘‘
*D free_abmon_inj_df          <f:free_abmon:E>.inj== free_abmon_inj{}(<f>)
*A free_abmon_inj             f.inj ==  f.2.1
*T free_abmon_inj_wf  0.5 sec.
⊢ ∀S:DSet. ∀f:FAbMon(S).  f.inj ∈ |S| ⟶ |f.mon|
|
BY ModulePiTac 3 ‘‘free_abmon_mon free_abmon_inj free_abmon_umap‘‘
*D free_abmon_umap_df         <f:free_abmon:E>.umap== free_abmon_umap{}(<f>)
*A free_abmon_umap            f.umap ==  f.2.2
*T free_abmon_umap_wf  0.5 sec.
⊢ ∀S:DSet. ∀f:FAbMon(S).
|    f.umap ∈ mon':AbMon ⟶ f':(|S| ⟶ |mon'|) ⟶ {!g:MonHom(f.mon,mon') | g o f.inj = f'}
|
BY ModulePiTac 3 ‘‘free_abmon_mon free_abmon_inj free_abmon_umap‘‘
*M create_free_abmonoid
               Class Declaration for: FAbMon(S)
                 Long Name: free_abmonoid
                 Short Name: free_abmon
                 Parameters:
                    S : DSet
                 Fields:
                    mon : AbMon
                    inj : |S| ⟶ |mon|
                    umap : mon':AbMon
                           ⟶ f':(|S| ⟶ |mon'|)
                           ⟶ {!g:MonHom(mon,mon') | g o inj = f'}
                 Universe: 𝕌'
*T free_abmon_umap_properties  17.7 sec.
⊢ ∀S:DSet. ∀M:FAbMon(S). ∀N:AbMon. ∀p:|S| ⟶ |N|.
|    M.umap N p o M.inj = p ∧ (∀f:MonHom(M.mon,N). f o M.inj = p ⇒ f = M.umap N p)
|
BY (UnivCD ...a)
|
1. S: DSet
2. M: FAbMon(S)
3. N: AbMon
4. p: |S| ⟶ |N|
⊢ M.umap N p o M.inj = p ∧ (∀f:MonHom(M.mon,N). f o M.inj = p ⇒ f = M.umap N p)
|
BY D 2 THEN D 3
|  THEN AbReduce 0
|
2. mon: AbMon
3. inj: |S| ⟶ |mon|
4. M2: mon':AbMon ⟶ f':(|S| ⟶ |mon'|) ⟶ {!g:MonHom(mon,mon') | g o inj = f'}
5. N: AbMon
6. p: |S| ⟶ |N|
⊢ M2 N p o inj = p ∧ (∀f:MonHom(mon,N). f o inj = p ⇒ f = M2 N p)
|
BY (Assert ⌜M2 N p ∈ {!g:MonHom(mon,N) | g o inj = p}⌝ ...a)
|
7. M2 N p ∈ {!g:MonHom(mon,N) | g o inj = p}
|
BY (MemTypeHD (-1) ...a)
|
7. M2 N p = M2 N p
[8]. M2 N p o inj = p ∧ (∀y:MonHom(mon,N). y o inj = p ⇒ y = M2 N p)
```

```
|
BY (GenUnivCD ...a)
|\
| 8. M2 N p o inj = p ∧ (∀y:MonHom(mon,N). y o inj = p ⇒ y = M2 N p)
| ⊢ M2 N p o inj = p
| |
1 BY (BHyp (-1) ...)
 \
  8. M2 N p o inj = p ∧ (∀y:MonHom(mon,N). y o inj = p ⇒ y = M2 N p)
  9. f: MonHom(mon,N)
  10. f o inj = p
  ⊢ f = M2 N p
  |
  BY (BHyp (-3) ...)
*T free_abmon_umap_properties_1  11.9 sec.
⊢ ∀S:DSet. ∀M:FAbMon(S). ∀N:AbMon. ∀p:|S| ⟶ |N|.  M.umap N p o M.inj = p
|
BY (UnivCD ...a)
|
1. S: DSet
2. M: FAbMon(S)
3. N: AbMon
4. p: |S| ⟶ |N|
⊢ M.umap N p o M.inj = p
|
BY D 2 THEN D 3
|  THEN AbReduce 0
|
2. mon: AbMon
3. inj: |S| ⟶ |mon|
4. M2: mon':AbMon ⟶ f':(|S| ⟶ |mon'|) ⟶ {!g:MonHom(mon,mon') | g o inj = f'}
5. N: AbMon
6. p: |S| ⟶ |N|
⊢ M2 N p o inj = p
|
BY (Assert ⌜M2 N p ∈ {!g:MonHom(mon,N) | g o inj = p}⌝ ...a)
|
7. M2 N p ∈ {!g:MonHom(mon,N) | g o inj = p}
|
BY (MemTypeHD (-1) ...a)
|
7. M2 N p = M2 N p
8. M2 N p o inj = p ∧ (∀y:MonHom(mon,N). y o inj = p ⇒ y = M2 N p)
|
BY (GenUnivCD ...a)
|
|
BY (BHyp (-1) ...)
*T free_abmon_endomorph_is_id  7.4 sec.
⊢ ∀S:DSet. ∀M:FAbMon(S). ∀f:MonHom(M.mon,M.mon).  f o M.inj = M.inj ⇒ f = Id{|M.mon|}
|
BY (UnivCD ...a)
|
1. S: DSet
2. M: FAbMon(S)
3. f: MonHom(M.mon,M.mon)
4. f o M.inj = M.inj
```

```
⊢ f = Id{|M.mon|}
|
BY % Should be nicer way to get this...%
|
|  (InstLemma ‘free_abmon_umap_properties‘
|    [⌈S⌉;⌈M⌉;⌈M.mon⌉;⌈M.inj⌉]
|    THENM D (-1) ...a)
|
5. M.umap M.mon M.inj o M.inj = M.inj
6. ∀f:MonHom(M.mon,M.mon). f o M.inj = M.inj ⇒ f = M.umap M.mon M.inj
|
BY (InstHyp [⌈Id{|M.mon|}⌉] 6
|     ...a)
|\
| ⊢ Id{|M.mon|} o M.inj = M.inj
| |
1 BY (RW CompIdNormC 0  ...)
 \
  7. Id{|M.mon|} = M.umap M.mon M.inj
  |
  BY (FHyp 6 [4] ...)
*T free_abmon_unique  32.4 sec.
⊢ ∀S:DSet. ∀M,N:FAbMon(S).
|    ∃f:MonHom(M.mon,N.mon). ∃g:MonHom(N.mon,M.mon). InvFuns(|M.mon|;|N.mon|;f;g)
|
BY (UnivCD ...a)
|
1. S: DSet
2. M: FAbMon(S)
3. N: FAbMon(S)
⊢ ∃f:MonHom(M.mon,N.mon). ∃g:MonHom(N.mon,M.mon). InvFuns(|M.mon|;|N.mon|;f;g)
|
BY (InstConcl  [⌈M.umap N.mon N.inj⌉;⌈N.umap M.mon M.inj⌉]
|   THENM D 0 ...a)
|\
| ⊢ N.umap M.mon M.inj o M.umap N.mon N.inj = Id{|M.mon|}
| |
1 BY (BLemma ‘free_abmon_endomorph_is_id‘ ...a)
| |
| ⊢ (N.umap M.mon M.inj o M.umap N.mon N.inj) o M.inj = M.inj
| |
1 BY (RW CompIdNormC 0
|      THENM RewriteWith [] ‘‘free_abmon_umap_properties_1‘‘ 0 ...)
 \
  ⊢ M.umap N.mon N.inj o N.umap M.mon M.inj = Id{|N.mon|}
  |
  BY (BLemma ‘free_abmon_endomorph_is_id‘ ...a)
  |
  ⊢ (M.umap N.mon N.inj o N.umap M.mon M.inj) o N.inj = N.inj
  |
  BY (RW CompIdNormC 0
       THENM RewriteWith [] ‘‘free_abmon_umap_properties_1‘‘ 0 ...)
*T mk_fabmon  54.3 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀i:|s| → |g|. ∀U:g':AbMon → (|s| → |g'|) → |g| → |g'|.
|    (∀g':AbMon. ∀f:|s| → |g'|.
|       IsMonHom{g,g'}(U g' f)
|       ∧ U g' f o i = f
```

```
|       ∧ (∀u:|g| ⟶ |g'|. IsMonHom{g,g'}(u) ⇒ u o i = f ⇒ u = U g' f))
|    ⇒ <g, i, U> ∈ FAbMon(s)
|
BY (Unfold `free_abmonoid` 0 ...)
|
1. s: DSet
2. g: AbMon
3. i: |s| ⟶ |g|
4. U: g':AbMon ⟶ (|s| ⟶ |g'|) ⟶ |g| ⟶ |g'|
5. ∀g':AbMon. ∀f:|s| ⟶ |g'|.
     IsMonHom{g,g'}(U g' f)
     ∧ U g' f o i = f
     ∧ (∀u:|g| ⟶ |g'|. IsMonHom{g,g'}(u) ⇒ u o i = f ⇒ u = U g' f)
⊢ U ∈ mon':AbMon ⟶ f':(|s| ⟶ |mon'|) ⟶ {!g:MonHom(g,mon') | g o i = f'}
|
BY % Can't use inclusion tactics here, since they
|    don't track hyp predicates such as hyp 5 here %
|  (Assert ⌜∀g':AbMon. ∀f:|s| ⟶ |g'|.  U g' f ∈ {!f':MonHom(g,g') | f' o i = f}⌝
|    ...a)
|\
| 6. g': AbMon
| 7. f: |s| ⟶ |g'|
| ⊢ U g' f ∈ {!f':MonHom(g,g') | f' o i = f}
| |
1 BY (MemTypeCD ...)
| |\
| | ⊢ U g' f ∈ MonHom(g,g')
| | |
1 2 BY (MemTypeCD THENP BHyp 5 ...)
| |\
| | ⊢ U g' f o i = f
| | |
1 2 BY (BHyp 5 ...)
|  \
|   8. y: MonHom(g,g')
|   9. y o i = f
|   ⊢ y = U g' f
|   |
1   BY (EqTypeCD THENP AddProperties 8 ...)
|   |
|   ⊢ y = U g' f
|   |
1   BY (BHyp 5 ...)
|   |
|   ⊢ IsMonHom{g,g'}(y)
|   |
1   BY (AddProperties 8 ...)
 \
  6. ∀g':AbMon. ∀f:|s| ⟶ |g'|.  U g' f ∈ {!f':MonHom(g,g') | f' o i = f}
  |
  BY (ExtWith [`g\''] [⌜g':AbMon ⟶ (|s| ⟶ |g'|) ⟶ |g| ⟶ |g'|⌝]
  |   THENM ExtWith [`f'] [⌜(|s| ⟶ |g'|) ⟶ |g| ⟶ |g'|⌝]  ...a)
  |
  7. g': AbMon
  8. f: |s| ⟶ |g'|
  ⊢ U g' f ∈ {!g:MonHom(g,g') | g o i = f}
  |
```

```
  BY (BHyp 6 ...)
*C mcopower_sig_com
               ==================
               MONOID COPOWER ADT
               ==================
*D mcopower_sig_df
               MCopowerSig{<i:level>}(<s:s:*>;<g:g:*>)== mcopower_sig{<i>:l}(<s>; <g>)
               MCopowerSig(<s:s:*>;<g:g:*>)== mcopower_sig{i:l}(<s>; <g>)
*A mcopower_sig
               MCopowerSig(s;g) ==
                 mon:AbMon
                 × inj:(|s| → |g| → |mon|)
                 × (h:AbMon → (|s| → |g| → |h|) → |mon| → |h|)
*T mcopower_sig_wf  5.0 sec.
⊢ ∀s:DSet. ∀g:AbMon.  MCopowerSig(s;g) ∈ 𝕌'
|
BY Unfold `mcopower_sig` 0 THEN Auto
*D mcopower_mon_df            <m:mcopower:E>.mon== mcopower_mon{}(<m>)
*A mcopower_mon               m.mon ==  m.1
*T mcopower_mon_wf  0.5 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).  m.mon ∈ AbMon
|
BY ModulePiTac 3 ``mcopower_mon mcopower_inj mcopower_umap``
*D mcopower_inj_df            <m:mcopower:E>.inj== mcopower_inj{}(<m>)
*A mcopower_inj               m.inj ==  m.2.1
*T mcopower_inj_wf  0.6 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).  m.inj ∈ |s| → |g| → |m.mon|
|
BY ModulePiTac 3 ``mcopower_mon mcopower_inj mcopower_umap``
*D mcopower_umap_df           <m:mcopower:E>.umap== mcopower_umap{}(<m>)
*A mcopower_umap              m.umap ==  m.2.2
*T mcopower_umap_wf  0.6 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).
|    m.umap ∈ h:AbMon → (|s| → |g| → |h|) → |m.mon| → |h|
|
BY ModulePiTac 3 ``mcopower_mon mcopower_inj mcopower_umap``
*M create_mcopower_sig
               Class Declaration for: MCopowerSig(s;g)
                 Long Name: mcopower_sig
                 Short Name: mcopower
                 Parameters:
                    s : DSet
                    g : AbMon
                 Fields:
                    mon : AbMon
                    inj : |s| → |g| → |mon|
                    umap : h:AbMon → (|s| → |g| → |h|) → |mon| → |h|
                 Universe: 𝕌'
*D mcopower_df MCopower{<i:level>}(<s:s:*>;<g:g:*>)== mcopower{<i>:l}(<s>; <g>)
               MCopower(<s:s:*>;<g:g:*>)== mcopower{i:l}(<s>; <g>)
*A mcopower    MCopower(s;g) ==
                 {c:MCopowerSig(s;g)|
                  (∀j:|s|. IsMonHom{g,c.mon}(c.inj j))
                  ∧ (∀h:AbMon. ∀f:|s| → MonHom(g,h).
                       c.umap h f = !v:|c.mon| → |h|
                                    IsMonHom{c.mon,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))}
*T mcopower_wf  13.6 sec.
```

```
⊢ ∀s:DSet. ∀g:AbMon.  MCopower(s;g) ∈ 𝕌'
|
BY (Unfold `mcopower` 0 ...)
*M mcopower_ml_inc
               add_set_inclusion_info
                 `mcopower` `mcopower_sig`
                 AbSetDForInc Auto ;;
*T mcopower_properties  20.9 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g).
|    (∀j:|s|. IsMonHom{g,c.mon}(c.inj j))
|    ∧ (∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
|         c.umap h f = !v:|c.mon| ⟶ |h|. IsMonHom{c.mon,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))
|
BY Try ProvePropertiesLemma
*T mcopower_inj_is_hom  1.1 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀j:|s|.  IsMonHom{g,c.mon}(c.inj j)
|
BY (RepD
    THENM AssertLemma  `mcopower_properties` []
    THENM Unfold `uni_sat` (-1)
    THENM BHyp (-1) ...)
*T mcopower_umap_is_hom  2.4 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
|    IsMonHom{c.mon,h}(c.umap h f)
|
BY (RepD
    THENM AssertLemma  `mcopower_properties` []
    THENM Unfold `uni_sat` (-1)
    THENM BHyp (-1) ...)
*T mcopower_umap_comm_tri  2.9 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h). ∀j:|s|.
|    f j = c.umap h f o c.inj j
|
BY (RepD
    THENM AssertLemma  `mcopower_properties` []
    THENM Unfold `uni_sat` (-1)
    THENM BHyp (-1) ...)
*T mcopower_umap_comm_tri_a  4.9 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h). ∀a:|g|. ∀j:|s|.
|    f j a = c.umap h f (c.inj j a)
|
BY (RepD ...a)
|
1. s: DSet
2. g: AbMon
3. c: MCopower(s;g)
4. h: AbMon
5. f: |s| ⟶ MonHom(g,h)
6. a: |g|
7. j: |s|
⊢ f j a = c.umap h f (c.inj j a)
|
BY RWH add_composeC 0
   THENM (RWO "mcopower_umap_comm_tri<" 0 ...)
*T mcopower_umap_unique  12.1 sec.
⊢ ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h). ∀a':|c.mon| ⟶ |h|.
|    IsMonHom{c.mon,h}(a') ⇒ (∀j:|s|. f j = a' o c.inj j) ⇒ a' = c.umap h f
```

```
|
BY (RepD
    THENM AssertLemma  ‘mcopower_properties‘ []
    THENM Unfold ‘uni_sat‘ (-1)
    THENM HypBackchain ...)
*C fabmon_of_nat_mcp_com
               ======================================
               CREATION OF FREE ABMON FROM MON COPOWER
               ======================================
*D fabmon_of_nat_mcp_df       fabmon_of_nat_mcp(<F:F:*>)== fabmon_of_nat_mcp{}(<F>)
*A fabmon_of_nat_mcp
               fabmon_of_nat_mcp(m) ==
                 <m.mon, λu.m.inj u zhgrp(1), λm',f.m.umap m' (λz,n.nat(n) · f z)>
*T fabmon_of_nat_mcp_wf  61.9 sec.
⊢ ∀s:DSet. ∀m:MCopower(s;<ℤ+>↓hgrp).  fabmon_of_nat_mcp(m) ∈ FAbMon(s)
|
BY (Unfold ‘fabmon_of_nat_mcp‘ 0
|   THEN RepD
|   THENM BLemma ‘mk_fabmon‘
|   THENM GenRepD ...a)
|\
| 1. s: DSet
| 2. m: MCopower(s;<ℤ+>↓hgrp)
| 3. g': AbMon
| 4. f: |s| ⟶ |g'|
| ⊢ IsMonHom{m.mon,g'}((λm',f.m.umap m' (λz,n.nat(n) · f z)) g' f)
| |
1 BY (Reduce 0 THENM BLemma ‘mcopower_umap_is_hom‘ ...)
| |
| 5. z: |s|
| ⊢ (λn.nat(n) · f z) ∈ MonHom(<ℤ+>↓hgrp,g')
| |
1 BY (MemTypeCD THENP BLemma ‘zhgrp_op_mon_hom_1‘ ...)
|\
| 1. s: DSet
| 2. m: MCopower(s;<ℤ+>↓hgrp)
| 3. g': AbMon
| 4. f: |s| ⟶ |g'|
| ⊢ (λm',f.m.umap m' (λz,n.nat(n) · f z)) g' f o (λu.m.inj u zhgrp(1)) = f
| |
1 BY (Ext THEN Reduce 0 ...)
| |
| 5. x: |s|
| ⊢ m.umap g' (λz,n.nat(n) · f z) (m.inj x zhgrp(1)) = f x
| |
1 BY (RWH add_composeC 0
| |   THENM RWO "mcopower_umap_comm_tri<" 0 ...a)
| |\
| | 6. z: |s|
| | ⊢ (λn.nat(n) · f z) ∈ MonHom(<ℤ+>↓hgrp,g')
| | |
1 2 BY (MemTypeCD THENP BLemma ‘zhgrp_op_mon_hom_1‘ ...)
|  \
|   ⊢ (λz,n.nat(n) · f z) x zhgrp(1) = f x
|   |
1   BY (Reduce 0 THENM RWO "mon_nat_op_one" 0 ...)
 \
```

```
1. s: DSet
2. m: MCopower(s;<ℤ+>↓hgrp)
3. g': AbMon
4. f: |s| ⟶ |g'|
5. u: |m.mon| ⟶ |g'|
6. IsMonHom{m.mon,g'}(u)
7. u o (λu.m.inj u zhgrp(1)) = f
⊢ u = (λm',f.m.umap m' (λz,n.nat(n) · f z)) g' f
|
BY (Reduce 0 THENM BLemma `mcopower_umap_unique` ...)
|\
| 8. z: |s|
| ⊢ (λn.nat(n) · f z) ∈ MonHom(<ℤ+>↓hgrp,g')
| |
1 BY (MemTypeCD THENP BLemma `zhgrp_op_mon_hom_1` ...)
 \
  8. j: |s|
  ⊢ (λz,n.nat(n) · f z) j = u o m.inj j
  |
  BY (Reduce 0
  |   THENM RWO "7<" 0
  |   THENM Ext
  |   THENM Reduce 0 ...a)
  |
  9. x: |<ℤ+>|+
  ⊢ nat(x) · u (m.inj j zhgrp(1)) = u (m.inj j x)
  |
  BY (RWW "mon_nat_op_hom_swap" 0 ...a)
  |\
  | ⊢ m.inj j ∈ MonHom(<ℤ+>↓hgrp,m.mon)
  | |
  1 BY (MemTypeCD THENP BLemma `mcopower_inj_is_hom` ...)
   \
    ⊢ u (m.inj j (nat(x) · zhgrp(1))) = u (m.inj j x)
    |
    BY % Ugggh %
    |  let C = EvalC ``mon_nat_op int_hgrp_el`` in
    |   RWH (MacroC `x` C ⌈a · zhgrp(b)⌉ C ⌈a · b⌉) 0
    |
    ⊢ u (m.inj j (nat(x) · 1)) = u (m.inj j x)
    |
    BY (RWW "nat_op_on_nat_add_mon" 0 ...a)
    |
    ⊢ u (m.inj j (nat(x) * 1)) = u (m.inj j x)
    |
    BY (RepeatM EqCD ...)
    |
    ⊢ nat(x) * 1 = x
    |
    BY (Eval ``int_hgrp_to_nat`` 0
    |   THENM InvertRel 0
    |   THENM EqTypeCD
    |   THENP AddProperties 9 ...)
    |
    ⊢ x = x * 1
    |
    BY (Reduce 0 ...)
```

```
*C fabgrp_com  ============================
               FREE ABELIAN GROUP OVER A SET
               ============================
*D fabgrp_sig_df
               FAbGrpSig{<i:level>}(<s:s:*>)== fabgrp_sig{<i>:l}(<s>)
               FAbGrpSig(<s:s:*>)== fabgrp_sig{i:l}(<s>)
*A fabgrp_sig  FAbGrpSig(s) ==
                 grp:AbGrp
                 × inj:(|s| → |grp|)
                 × (grp':AbGrp → (|s| → |grp'|) → |grp| → |grp'|)
*T fabgrp_sig_wf  3.1 sec.
⊢ ∀s:DSet. FAbGrpSig(s) ∈ 𝕌'
|
BY Unfold ‘fabgrp_sig‘ 0 THEN Auto
*D fabgrp_grp_df              <f:fabgrp:E>.grp== fabgrp_grp{}(<f>)
*A fabgrp_grp                 f.grp ==  f.1
*T fabgrp_grp_wf  0.4 sec.
⊢ ∀s:DSet. ∀f:FAbGrpSig(s).  f.grp ∈ AbGrp
|
BY ModulePiTac 3 ‘‘fabgrp_grp fabgrp_inj fabgrp_umap‘‘
*D fabgrp_inj_df              <f:fabgrp:E>.inj== fabgrp_inj{}(<f>)
*A fabgrp_inj                 f.inj ==  f.2.1
*T fabgrp_inj_wf  0.4 sec.
⊢ ∀s:DSet. ∀f:FAbGrpSig(s).  f.inj ∈ |s| → |f.grp|
|
BY ModulePiTac 3 ‘‘fabgrp_grp fabgrp_inj fabgrp_umap‘‘
*D fabgrp_umap_df             <f:fabgrp:E>.umap== fabgrp_umap{}(<f>)
*A fabgrp_umap                f.umap ==  f.2.2
*T fabgrp_umap_wf  0.4 sec.
⊢ ∀s:DSet. ∀f:FAbGrpSig(s).  f.umap ∈ grp':AbGrp → (|s| → |grp'|) → |f.grp| → |grp'|
|
BY ModulePiTac 3 ‘‘fabgrp_grp fabgrp_inj fabgrp_umap‘‘
*M create_fabgrp_sig
               Class Declaration for: FAbGrpSig(s)
                 Long Name: fabgrp_sig
                 Short Name: fabgrp
                 Parameters:
                    s : DSet
                 Fields:
                    grp : AbGrp
                    inj : |s| → |grp|
                    umap : grp':AbGrp → (|s| → |grp'|) → |grp| → |grp'|
                 Universe: 𝕌'
*D fabgrp_df   FAbGrp{<i:level>}(<s:s:*>)== fabgrp{<i>:l}(<s>)
               FAbGrp(<s:s:*>)== fabgrp{i:l}(<s>)
*A fabgrp      FAbGrp(s) ==
                 {fg:FAbGrpSig(s)|
                  ∀g:AbGrp. ∀h:|s| → |g|.
                    fg.umap g h = !h':|fg.grp| → |g|
                                    IsMonHom{fg.grp,g}(h') ∧ h' o fg.inj = h}
*T fabgrp_wf  6.2 sec.
⊢ ∀s:DSet. FAbGrp(s) ∈ 𝕌'
|
BY (Unfold ‘fabgrp‘ 0 ...)
*C gcopower_com
               =============
               GROUP COPOWER
```

```
              =============
              We need to create a new signature here, because the desired
              group copower type is incomparable with the monoid
              copower sig.  Maybe it is is a poor design decision to include
              any domain restrictions in signatures.
              I assume that Any total function with the class of monoids or groups
              as domain, will in practice be also total over the more
              general signatures.
*D gcopower_sig_df
              GCopowerSig{<i:level>}(<s:s:*>;<g:g:*>)== gcopower_sig{<i>:l}(<s>; <g>)
              GCopowerSig(<s:s:*>;<g:g:*>)== gcopower_sig{i:l}(<s>; <g>)
*A gcopower_sig
              GCopowerSig(s;g) ==
                grp:GrpSig
                × inj:(|s| → |g| → |grp|)
                × (h:AbGrp → (|s| → |g| → |h|) → |grp| → |h|)
*T gcopower_sig_wf  2.9 sec.
⊢ ∀s:PosetSig. ∀g:GrpSig.  GCopowerSig(s;g) ∈ 𝕌'
|
BY Unfold ‘gcopower_sig‘ 0 THEN Auto
*D gcopower_grp_df            <g1:gcopower:E>.grp== gcopower_grp{}(<g1>)
*A gcopower_grp               g1.grp ==  g1.1
*T gcopower_grp_wf  0.6 sec.
⊢ ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).  g1.grp ∈ GrpSig
|
BY ModulePiTac 3 ‘‘gcopower_grp gcopower_inj gcopower_umap‘‘
*D gcopower_inj_df            <g1:gcopower:E>.inj== gcopower_inj{}(<g1>)
*A gcopower_inj               g1.inj ==  g1.2.1
*T gcopower_inj_wf  0.6 sec.
⊢ ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).  g1.inj ∈ |s| → |g| → |g1.grp|
|
BY ModulePiTac 3 ‘‘gcopower_grp gcopower_inj gcopower_umap‘‘
*D gcopower_umap_df           <g1:gcopower:E>.umap== gcopower_umap{}(<g1>)
*A gcopower_umap              g1.umap ==  g1.2.2
*T gcopower_umap_wf  0.6 sec.
⊢ ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).
|    g1.umap ∈ h:AbGrp → (|s| → |g| → |h|) → |g1.grp| → |h|
|
BY ModulePiTac 3 ‘‘gcopower_grp gcopower_inj gcopower_umap‘‘
*M create_gcopower_sig
              Class Declaration for: GCopowerSig(s;g)
                Long Name: gcopower_sig
                Short Name: gcopower
                Parameters:
                   s : PosetSig
                   g : GrpSig
                Fields:
                   grp : GrpSig
                   inj : |s| → |g| → |grp|
                   umap : h:AbGrp → (|s| → |g| → |h|) → |grp| → |h|
                Universe: 𝕌'
*D mon_p_df                   mon_p(<g:g:*>)== mon_p{}(<g>)
*A mon_p                      mon_p(g) ==  IsMonoid(|g|;*;e)
*T mon_p_wf  0.6 sec.
⊢ ∀g:GrpSig. mon_p(g) ∈ ℙ
|
BY (Unfold ‘mon_p‘ 0 ...)
```

```
*D grp_p_df                    grp_p(<g:g:*>)== grp_p{}(<g>)
*A grp_p                       grp_p(g) ==  IsGroup(|g|;*;e;∼)
*T grp_p_wf  0.7 sec.
⊢ ∀g:GrpSig. grp_p(g) ∈ ℙ
|
BY (Unfold ‘grp_p‘ 0 ...)
*D rng_p_df                    rng_p(<r:r:*>)== rng_p{}(<r>)
*A rng_p                       rng_p(r) ==  grp_p(r↓+gp) ∧ mon_p(r↓xmn) ∧ BiLinear(|r|;+r;*)
*T rng_p_wf  1.4 sec.
⊢ ∀r:RngSig. rng_p(r) ∈ ℙ
|
BY (Unfold ‘rng_p‘ 0 ...)
*D alg_p_df                    AlgP(<r:rng:*>;<a:alg:*>)== alg_p{}(<r>; <a>)
*A alg_p       AlgP(r;a) ==
                 rng_p(a↓rg)
                 ∧ IsAction(|r|;*;1;|a|;·a)
                 ∧ IsBilinear(|r|;|a|;|a|;+r;+a;+a;·a)
                 ∧ (∀p:|r|. Dist1op2opLR(|a|;·a p;xa))
*T alg_p_wf  4.1 sec.
⊢ ∀r:RngSig. ∀a:AlgebraSig(|r|).  AlgP(r;a) ∈ ℙ
|
BY (Unfold ‘alg_p‘ 0 ...)
*D gcopower_df                 gcopower{<i:level>}(<s:s:*>;<g:g:*>)== gcopower{<i>:l}(<s>; <g>)
*A gcopower    gcopower{i}(s;g) ==
                 {c:GCopowerSig(s;g)|
                  IsEqFun(|c.grp|;=b)
                  ∧ grp_p(c.grp)
                  ∧ Comm(|c.grp|;*)
                  ∧ (∀j:|s|. IsMonHom{g,c.grp}(c.inj j))
                  ∧ (∀h:AbGrp. ∀f:|s| ⟶ MonHom(g,h).
                       c.umap h f = !v:|c.grp| ⟶ |h|
                                    IsMonHom{c.grp,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))}
*T gcopower_wf  16.1 sec.
⊢ ∀s:DSet. ∀g:AbGrp.  gcopower{i}(s;g) ∈ 𝕌’
|
BY (Unfold ‘gcopower‘ 0 ...)
*M gcopower_ml_inc
               add_set_inclusion_info
                 ‘gcopower‘ ‘gcopower_sig‘
                 AbSetDForInc Auto ;;
*T gcopower_properties  34.2 sec.
⊢ ∀s:DSet. ∀g:AbGrp. ∀c:gcopower{i}(s;g).
|    IsEqFun(|c.grp|;=b)
|    ∧ grp_p(c.grp)
|    ∧ Comm(|c.grp|;*)
|    ∧ (∀j:|s|. IsMonHom{g,c.grp}(c.inj j))
|    ∧ (∀h:AbGrp. ∀f:|s| ⟶ MonHom(g,h).
|         c.umap h f = !v:|c.grp| ⟶ |h|. IsMonHom{c.grp,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))
|
BY Try ProvePropertiesLemma
|
1. s: DSet
2. g: AbGrp
3. c: GCopowerSig(s;g)
[4]. IsEqFun(|c.grp|;=b)
     ∧ grp_p(c.grp)
     ∧ Comm(|c.grp|;*)
```

```
     ∧ (∀j:|s|. IsMonHom{g,c.grp}(c.inj j))
     ∧ (∀h:AbGrp. ∀f:|s| → MonHom(g,h).
          c.umap h f = !v:|c.grp| → |h|. IsMonHom{c.grp,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))
⊢ SqStable(grp_p(c.grp))
|
BY Unfold ‘grp_p‘ 0 THEN Auto
*C fmonalg_com ====================
               FREE MONOID ALGEBRA
               ====================
*D fma_sig_df  FMASig{<i:level>}(<G:grp:*>;<A:rng:*>)== fma_sig{<i>:1}(<G>; <A>)
               FMASig(<G:grp:*>;<A:rng:*>)== fma_sig{i:1}(<G>; <A>)
*A fma_sig     FMASig(G;A) ==
                 alg:A-Algebra
                 × inj:(|G| → |alg|)
                 × (N:A-Algebra → (|G| → |N|) → |alg| → |N|)
*T fma_sig_wf  3.7 sec.
⊢ ∀G:GrpSig. ∀A:RngSig.  FMASig(G;A) ∈ 𝕌’
|
BY Unfold ‘fma_sig‘ 0 THEN Auto
*D fma_alg_df                <f:fma:E>.alg== fma_alg{}(<f>)
*A fma_alg                   f.alg ==  f.1
*T fma_alg_wf  0.5 sec.
⊢ ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).  f.alg ∈ A-Algebra
|
BY ModulePiTac 3 ‘‘fma_alg fma_inj fma_umap‘‘
*D fma_inj_df                <f:fma:E>.inj== fma_inj{}(<f>)
*A fma_inj                   f.inj ==  f.2.1
*T fma_inj_wf  0.6 sec.
⊢ ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).  f.inj ∈ |G| → |f.alg|
|
BY ModulePiTac 3 ‘‘fma_alg fma_inj fma_umap‘‘
*D fma_umap_df                <f:fma:E>.umap== fma_umap{}(<f>)
*A fma_umap                   f.umap ==  f.2.2
*T fma_umap_wf  0.7 sec.
⊢ ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).  f.umap ∈ N:A-Algebra → (|G| → |N|) → |f.alg| → |N|
|
BY ModulePiTac 3 ‘‘fma_alg fma_inj fma_umap‘‘
*M create_fma_sig
               % Create type of free (G,A) Monoid Algebras %
               % Do in two stages. Allows verification of existence
               of instances also to be broken up into stages
               Note that this type is more general than one with
               AlgebraSig(|A|) as the first domain of umap.
               This generality is necessary. It turns out that a umap
               considered in the polynom_3 theory isn’t even well-formed
               for arbitrary elements of AlgebraSig(|A|). (Problem reduces
               to that mset_for requires for functionality reasons the addition
               of the algebra to be an abelian monoid.    %
               Class Declaration for: FMASig(G;A)
                 Long Name: fma_sig
                 Short Name: fma
                 Parameters:
                    G : GrpSig
                    A : RngSig
                 Fields:
                    alg : AlgebraSig(|A|)
                    inj : |G| → |alg|
```

```
                  umap : N:A-Algebra → (|G| → |N|) → |alg| → |N|
                Universe: 𝕌'
*D fmonalg_df  FMonAlg{<i:level>}(<g:g:*>;<a:a:*>)== fmonalg{<i>:l}(<g>; <a>)
               FMonAlg(<g:g:*>;<a:a:*>)== fmonalg{i:l}(<g>; <a>)
*A fmonalg     FMonAlg(g;a) ==
                 {m:FMASig(g;a)|
                  IsMonHom{g,m.alg↓rg↓xmn}(m.inj)
                  ∧ (∀n:a-Algebra. ∀f:MonHom(g,n↓rg↓xmn).
                       m.umap n f = !f':|m.alg| → |n|
                                    IsAlgHom{a,m.alg,n}(f') ∧ f' o m.inj = f)}
*T fmonalg_wf  15.3 sec.
⊢ ∀g:AbMon. ∀a:CRng.  FMonAlg(g;a) ∈ 𝕌'
|
BY (Unfold `fmonalg` 0 ...)
*M fmonalg_ml_inc
               add_set_inclusion_info
                 `fmonalg` `fma_sig`
                 AbSetDForInc Auto ;;
*T fmonalg_properties  22.2 sec.
⊢ ∀g:AbMon. ∀a:CRng. ∀m:FMonAlg(g;a).
|    IsMonHom{g,m.alg↓rg↓xmn}(m.inj)
|    ∧ (∀n:a-Algebra. ∀f:MonHom(g,n↓rg↓xmn).
|         m.umap n f = !f':|m.alg| → |n|. IsAlgHom{a,m.alg,n}(f') ∧ f' o m.inj = f)
|
BY Try ProvePropertiesLemma
*D polynom_alg_df
               PolynomAlg{<i:level>}(<S:S:*>;<A:A:*>)== polynom_alg{<i>:l}(<S>; <A>)
               PolynomAlg(<S:S:*>;<A:A:*>)== polynom_alg{i:l}(<S>; <A>)
*A polynom_alg                PolynomAlg(S;A) ==  mo:FAbMon(S) × FMonAlg(mo.mon;A)
*T polynom_alg_wf  0.7 sec.
⊢ ∀S:DSet. ∀A:CRng.  PolynomAlg(S;A) ∈ 𝕌'
|
BY Unfold `polynom_alg` 0 THEN Auto
*D polyalg_mo_df              <p:polyalg:E>.mo== polyalg_mo{}(<p>)
*A polyalg_mo                 p.mo ==  p.1
*T polyalg_mo_wf  0.5 sec.
⊢ ∀S:DSet. ∀A:CRng. ∀p:PolynomAlg(S;A).  p.mo ∈ FAbMon(S)
|
BY ModulePiTac 2 ``polyalg_mo polyalg_poly``
*D polyalg_poly_df            <p:polyalg:E>.poly== polyalg_poly{}(<p>)
*A polyalg_poly               p.poly ==  p.2
*T polyalg_poly_wf  0.5 sec.
⊢ ∀S:DSet. ∀A:CRng. ∀p:PolynomAlg(S;A).  p.poly ∈ FMonAlg(p.mo.mon;A)
|
BY ModulePiTac 2 ``polyalg_mo polyalg_poly``
*M create_polynom_alg
               Class Declaration for: PolynomAlg(S;A)
                 Long Name: polynom_alg
                 Short Name: polyalg
                 Parameters:
                    S : DSet
                    A : CRng
                 Fields:
                    mo : FAbMon(S)            % the monomials %
                    poly : FMonAlg(mo.mon;A)  % the polynomials %
                 Universe: 𝕌'
*D rng_from_grp_df
```

```
              rng_from_grp(<g:g:*>;<times:times:*>;<one:one:*>;<div:div:*>)
              == rng_from_grp{}(<g>; <times>; <one>; <div>)
*A rng_from_grp
              rng_from_grp(g;times;one;div) ==  <|g|, =b, ≤b, *, e, ∼, times, one, div>
*T rng_from_grp_wf  8.7 sec.
⊢ ∀g:GrpSig. ∀times:|g| → |g| → |g|. ∀one:|g|. ∀div:|g| → |g| → ?|g|.
|    rng_from_grp(g;times;one;div) ∈ RngSig
|
BY (Unfolds ``rng_from_grp rng_sig`` 0 ...)
*D alg_from_rng_df
              alg_from_rng(<A:A:*>;<r:r:*>;<act:act:*>)== alg_from_rng{}(<A>; <r>; <act>)
*A alg_from_rng               alg_from_rng(A;r;act) ==  <|r|, =b, ≤b, +r, 0, -r, *, 1, ÷r, act>
*T alg_from_rng_wf  10.7 sec.
⊢ ∀A:𝕌. ∀r:RngSig. ∀act:A → |r| → |r|.  alg_from_rng(A;r;act) ∈ AlgebraSig(A)
|
BY (Unfolds ``alg_from_rng algebra_sig`` 0 ...)
*D fma_from_gcp_df
              fma_from_gcp(<g:g:*>;<r:r:*>;<c:c:*>;<a:a:*>)
              == fma_from_gcp{}(<g>; <r>; <c>; <a>)
*A fma_from_gcp
              fma_from_gcp(g;r;c;a) ==
                <a
                , λp:|g|. c.inj p 1
                , λa.(λh:|g| → |a|. c.umap a↓grp (λp:|g|. λv:|r|. ·a v (h p)))>
*T fma_from_gcp_wf  17.0 sec.
⊢ ∀g:GrpSig. ∀r:RngSig. ∀c:GCopowerSig(g↓set;r↓+gp). ∀a:r-Algebra.
|    a↓grp = c.grp ⇒ fma_from_gcp(g;r;c;a) ∈ FMASig(g;r)
|
BY (Unfolds ``fma_sig fma_from_gcp`` 0
|   THENM RepD
|   THENM MemCD THENML [Id;MemCD] ...a)
|\
| 1. g: GrpSig
| 2. r: RngSig
| 3. c: GCopowerSig(g↓set;r↓+gp)
| 4. a: r-Algebra
| 5. a↓grp = c.grp
| ⊢ a ∈ r-Algebra
| |
1 BY Auto
|\
| 1. g: GrpSig
| 2. r: RngSig
| 3. c: GCopowerSig(g↓set;r↓+gp)
| 4. a: r-Algebra
| 5. a↓grp = c.grp
| ⊢ (λp:|g|. c.inj p 1) ∈ |g| → |a|
| |
1 BY (MemCD ...a)
| |
| 6. p: |g|
| ⊢ c.inj p 1 ∈ |a|
| |
1 BY RWH grp_of_moduleC 0
| |  THENM (RWTH (HypC 5) 0 ...a)
| |
| ⊢ c.inj p 1 ∈ |c.grp|
```

```
| |
1 BY Auto
 \
  1. g: GrpSig
  2. r: RngSig
  3. c: GCopowerSig(g↓set;r↓+gp)
  4. a: r-Algebra
  5. a↓grp = c.grp
  ⊢ (λa.(λh:|g| ⟶ |a|. c.umap a↓grp (λp:|g|. λv:|r|. ·a v (h p)))) ∈ N:r-Algebra
  |                                                                    ⟶ (|g| ⟶ |N|)
  |                                                                    ⟶ |a|
  |                                                                    ⟶ |N|
  |
  BY (RepeatMFor 2 MemCD ...a)
  |
  6. a1: r-Algebra
  7. h: |g| ⟶ |a1|
  ⊢ c.umap a1↓grp (λp:|g|. λv:|r|. ·a1 v (h p)) ∈ |a| ⟶ |a1|
  |
  BY (RWH grp_of_moduleC 0
  |   THENM RWTH (HypC 5) 0  ...a)
  |
  ⊢ c.umap a1↓grp (λp:|g|. λv:|r|. ·a1 v (h p)) ∈ |c.grp| ⟶ |(a1↓grp)|
  |
  BY Auto
#T fma_from_gcp_wf2  23.2 sec.
⊢ ∀g:Mon. ∀r:CRng. ∀c:gcopower{i}(g↓set;r↓+gp). ∀a:r-Algebra.
|    a↓grp = c.grp ⇒ fma_from_gcp(g;r;c;a) ∈ FMonAlg(g;r)
|
BY (RepD THENM MemTypeCD ...)
|  THEN Reduce 0
|\
| 1. g: Mon
| 2. r: CRng
| 3. c: gcopower{i}(g↓set;r↓+gp)
| 4. a: r-Algebra
| 5. a↓grp = c.grp
| ⊢ IsMonHom{g,a↓rg↓xmn}(λp:|g|. c.inj p 1)
| |
| INCOMPLETE
 \
  1. g: Mon
  2. r: CRng
  3. c: gcopower{i}(g↓set;r↓+gp)
  4. a: r-Algebra
  5. a↓grp = c.grp
  6. n: r-Algebra
  7. f: MonHom(g,n↓rg↓xmn)
  ⊢ c.umap n↓grp (λp:|g|. λv:|r|. ·n v (f p)) = !f':|a| ⟶ |n|
  |                                                 IsAlgHom{r,a,n}(f')
  |                                                 ∧ f' o (λp:|g|. c.inj p 1) = f
  |
  INCOMPLETE
*C polynom_1_end              ************************************
```